ARMY RESEARCH LABORATORY

Acoustic and Seismic Sensor Placement (ASSP) Application Programming Interface (API) Version 1.0

by David Marlin and Shane Thomas

ARL-TR-4432 April 2008

Army Research Laboratory
White Sands Missile Range, NM 88002- 5501

ARL-TR-4432 **April 2008**

Acoustic and Seismic Sensor Placement (ASSP) Application Programming Interface (API) Version 1.0

David Marlin
Computational and Information Sciences Directorate, ARL

and

Shane Thomas
Physical Science Laboratory, New Mexico State University

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE *(DD-MM-YYYY)* April 2008	**2. REPORT TYPE** Final	**3. DATES COVERED (From - To)**
4. TITLE AND SUBTITLE Acoustic and Seismic Sensor Placement (ASSP) Application Programming Interface (API) Version 1.0		**5a. CONTRACT NUMBER**
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) David Marlin (ARL) and Shane Thomas (PSL)		**5d. PROJECT NUMBER**
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) U.S. Army Research Laboratory Computational and Information Sciences Directorate Battlefield Environment Division (ATTN: AMSRD-ARL-CI-ES) White Sands Missile Range, NM 88002-5501		**8. PERFORMING ORGANIZATION REPORT NUMBER** ARL-TR-4432
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		**10. SPONSOR/MONITOR'S ACRONYM(S)**
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
In this report, an Acoustic and Seismic Sensor Placement (ASSP) application programming interface (API) for acoustic sensor placement is described. The API is implemented in both C++ and Java, with functionality for the later provided through the former via the Java Native Interface (JNI). This API is based on the Sensor Performance for Battlefield Environments (SPEBE) API, but the classes are not extensions of the SPEBE API. Instead, they include SPEBE objects as encapsulated data, accessible only through the methods. This serves the dual purpose of allowing a more natural interface dedicated to sensor placement, while at the same time protecting the user from inadvertent misuse of the more general SPEBE API. The ASSP interface includes classes to define the environment, including atmospheric, elevation, and ground characterization, define sensor locations and characteristics, compute the detection probability of these sensors for specified source type, and manage the resulting data as both a regularly-spaced grid and as a set of detection-probability contours.

15. SUBJECT TERMS
Acoustic sensors, sensor placement, acoustic propagation, acoustic performance modeling, acoustic decision aid

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** David Marlin
a. REPORT U	**b. ABSTRACT** U	**c. THIS PAGE** U	UU	44	**19b. TELEPHONE NUMBER** (*Include area code*) (575) 678-1524

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

Contents

List of Tables

Summary

The Sensor Performance Evaluator for Battlefield Environments (SPEBE) application programming interface (API) provides an outstanding suite of computational tools for modeling the performance of acoustic sensors in a wide range of operational environments and development situations. This includes the development of acoustic sensor placement decision aids to assist mission planners in determining the optimum distribution of sensors for a given detection objective. Unfortunately, the use of the SPEBE API requires a detailed knowledge of the SPEBE architecture and significant expertise in acoustic propagation, acoustic signatures and sensors, and meteorology.

The purpose of the Acoustic and Seismic Sensor Placement (ASSP) API is to present the programmer with a subset of SPEBE capabilities tailored to the specific needs of an operational decision aid. The objectives of the API are to simplify the process of setting up and generating required calculations while at the same time reducing the possibility of error resulting from improper configuration and sequencing of SPEBE calculations. To accomplish both objectives, the API encapsulates the SPEBE objects rather than extending the SPEBE classes. This allows the ASSP API complete flexibility in its class design while simultaneously preventing the direct manipulation of SPEBE methods and data.

The API is based on five classes centered on definition of the acoustic and seismic environment, estimation of the effects of this environment on acoustic and seismic propagation, definition of acoustic and seismic sensor configurations, prediction of acoustic and seismic detection probabilities, and representation of the resulting data. These classes are described in detail in this report.

INTENTIONALLY LEFT BLANK.

1. Introduction

The Acoustic and Seismic Sensor Placement (ASSP) Application Programmer Interface (API) provides the tools to develop acoustic sensor placement decision aids[*]. It is built upon the Sensor Performance Evaluator for Battlefield Environments (SPEBE) API[1], tailored to the specific needs of acoustic sensor placement planning. While the SPEBE API provides a general set of classes to invoke all the acoustic propagation and sensor performance modeling capabilities of SPEBE, ASSP provides a smaller set of classes, of reduced complexity, designed specifically to support sensor placement planning. Thus, the user of the ASSP API can concentrate on the development of the sensor placement decision aid without the need to master the full range of the SPEBE API, which requires far more expertise in acoustics and meteorology.

The API is implemented in both C++ and Java, with the functionality of the Java implementation provided through the C++ implementation via the Java Native Interface (JNI).

At the heart of sensor placement planning is the calculation of probability of detection (PD), which is dependent upon the propagation environment, type and locations of sensors, and acoustic and seismic signatures of sources and backgrounds. Thus, the API is set up to streamline the description of the environment, sources, backgrounds, and sensors, to invoke PD calculations, and to represent the resulting data as both regularly spaced grids and contours.

This is done via the five ASSP classes, which are not extensions of the SPEBE API classes, but rather encapsulate them and provide the methods necessary to manage them internally. The five classes are organized to facilitate the sensor placement tasks, and insulate the user from the more general, and therefore complex, classes of the SPEBE API.

1.1 Four Steps of Detection-Probability Prediction

The overall process of sensor detection-probability prediction can be divided into four steps:

1. Definition of the propagation environment, including the following:
    - coordinates, time zone, and terrain elevations of the region of interest
    - predominant ground characterization of the region of interest
    - predominant meteorological conditions in the region of interest

[*]It was originally developed to support the Networked Sensors for the Future Force (NSFF) Advanced Technology Demonstration, hence the name SPEBE4NSFF.

[1]Marlin, D.; Thomas, S. *Sensor Performance Evaluator for Battlefield Environments (SPEBE) C++ Application Programming Interface (API) Version 1.0*; ARL-TR-4363; U.S. Army Research Laboratory: White Sands Missile Range, NM, December 2007.

2. Estimation of acoustic and seismic propagation effects, depending on the environmental characterization and the target and sensor heights (but not the specific type of target or sensor)
3. Definition of the sensor configuration, including number, type, and location of a set of sensors
4. Prediction of detection probabilities, depending on the following:
   - the environment defined in step 1
   - the propagation effects defined and generated in step 2
   - the sensor set defined in step 3
   - the specific type of target of interest (but not its location)
   - the selection of active sensors within the sensor set

1.2 Five Classes of ASSP

ASSP includes five classes: four to perform the four steps defined above, and a fifth to encapsulate the detection probability data generated by step 4. The data can be retrieved from the fifth class either as a two-dimensional (2-D) grid or as a set of contours. The five classes (in order of the steps) are as follows:

1. **SPEBE_Environment**
2. **SPEBE_PropagationTable**
3. **SPEBE_SensorSet**
4. **SPEBE_PDCalculator**
5. **ResultGrid**

Refer to the documentation in section 3 of each class for a more detailed description of the class and its methods.

1.3 Miscellaneous

There are several enumerated types associated with the five classes. Refer to section 5.1 or particular class documentation for details.

There are also a few *dll* housekeeping functions which must be performed, including initialization and termination. Refer to section 5.2 for details.

Attention: Use of this library requires the definition of the ABFAHome environment variable, which must point to the directory in which ASSP is installed.

Class constructors and methods that take filename arguments will throw char* exceptions if ABFAHome isn't defined or files aren't found.

Each of the five classes had a destructor, which frees a number of internal arrays allocated within the *dll*. It is important to *delete* all objects when they are no longer needed. Failure to adhere to this practice may result in memory failure and crashes.

The source and sensor types described in section 5.1 can be easily expanded or modified. Currently they are a minimal set used for the initial development of the API.

1.4 Some Examples

1.4.1 Initialization and Shutdown

Every session must begin with the call

```
SPEBE_Initialize();
```

and should end with the call

```
SPEBE_Shutdown();
```

1.4.2 Environment Definition

```
// instantiate a new SPEBE_Environment, and set up the domain,
// ground type, seismic type, meteorology
SPEBE_Environment* theEnvironment = new SPEBE_Environment();
theEnvironment->SetDomain(33.0265, 33.1265, -106.239, -106.139,
        "f:\\", 6);
theEnvironment->SetGroundType(Sand);
theEnvironment->SetSeismicType(Desert);
theEnvironment->SetMeteorology(temp, rh, windSpd, windDir, year, day,
time, cloudCover);
```

1.4.3 Propagation Table Management

```
// instantiate a new SPEBE_PropagationTable, and generate
// a propagation table for an UGS3 sensor and Tracked source
SPEBE_PropagationTable* thePropTable = new
        SPEBE_PropagationTable(theEnvironment,
        "d:\\spebe4nsff\\spebe4nsff_develop\\debug", "NSFF_Test");
thePropTable->GenerateTable(Human, Tracked);
```

1.4.4 Sensor Definition

```
// instantiate a SPEBE_SensorSet for two Human and one Acoustic
// sensors, and set their coordinates
SPEBE_SensorSet* theSensors = new SPEBE_SensorSet(3, Human);
theSensors->ChangeSensorType(1, Acoustic);
theSensors->SetSensorLocation(0, 33.045, -106.255);
theSensors->SetSensorLocation(1, 33.045, -106.260);
theSensors->SetSensorLocation(2, 33.046, -106.260);
```

1.4.5 Detection Probability Calculations

```
// instantiate a SPEBE_PDCalculator for the propagation tables
// and sensors defined above, for a Tracked target, in
// LightBattle background noise,
// activate all sensors, and calculate their detection probability
SPEBE_PDCalculator* thePDCalculator = new
        SPEBE_PDCalculator(thePropTable,
        theSensors, Tracked, LightBattle);
thePDCalculator->Activate(0);
thePDCalculator->Activate(1);
thePDCalculator->Activate(2);
ResultGrid* theResults3 = thePDCalculator->GetPD();

// deactivate one sensor and calculate the new detection probability
thePDCalculator->Deactivate(2);
ResultGrid* theResults2 = thePDCalculator->GetPD();
```

2. Class List

Table 1 shows the Acoustic Sensor Placement Decision Aid API Class List, which includes the classes, structs, unions, and interfaces with brief descriptions.

Table 1. Class list.

Name	Description
ResultGrid	Encapsulates 2-D gridded data returned by SPEBE calculations, such as detection probability. Also generates contour plots of the encapsulated data.
SPEBE_Environment	Describes the general propagation environment of the region-of-interest, which encompasses the following: • domain (**SetDomain**) • coordinates of lower-left and upper-right corners of region-of-interest • time zone of lower-left corner • terrain elevations of region-of-interest • predominant ground type of region (**SetGroundType**, see also **SPEBE_GroundType**) • predominant seismological characteristics of region (**SetSeismicType**, see also **SPEBE_SeismicType**) • predominant meteorology of region (**SetMeteorology**, see also **TranslateCloudCover**)
SPEBE_PDCalculator	Generates detection probabilities for a *fixed* • environment (region and weather: **SPEBE_Environment**), • background noise level (**SPEBE_NoiseType**), • target type (**SPEBE_TargetType**), height, and direction, and • set of sensors (**SPEBE_SensorSet**).
SPEBE_PropagationTable	Generates and manages the propagation tables for a given environment (**SPEBE_Environment**). These tables are used by the **SPEBE_PDCalculator** to calculate detection probabilities. If you want to avoid errors leading to invalid results, **make sure you read and understand the documentation** for this class.
SPEBE_SensorSet	Manages a set of sensors chosen from **SPEBE_SensorType**.
SPEBE_TargetSet	Manages a set of targets, or acoustic/seismic sources, chosen from **SPEBE_TargetType**.

3. Class Descriptions

3.1 ResultGrid Class

The **ResultGrid** class encapsulates the 2-D gridded data returned by SPEBE sensor performance calculations, such as detection probability. It also generates contour plots of the encapsulated data to keep track of the computational grid coordinates, so that the data can be referenced to its location within the computational grid for contour generation. See also **PerformanceCalculator**.

Table 2 provides the public member functions for the **ResultGrid** class.

Table 2. Public member functions **ResultGrid** class.

Function	Description
	ResultGrid (**ResultGrid** *theArray) Copy constructor.
	~ResultGrid (void) The destructor. Always delete ResultGrid objects when they are no longer needed.
	ResultGrid (double *theData, int numRows, int numCols, CDomain *theDomain, const char *arrayName) Instantiate a new ResultGrid with data contained in a linear double* array.
void	**GenerateContours** (double contourValue) Generate a set of contours for the encapsulated data, for the specified *contourValue*.
int	**GetNumContours** (void) Returns the number of contours genereated by **ResultGrid::GenerateContours**.
int	**GetContourLength** (int contour) Returns the length of a specified contour generated by **ResultGrid::GenerateContours**.
double *	**GetContourLat** (int contour) Returns the latitudes of a specified contour generated by GenerateContours.
double *	**GetContourNorthing** (int contour) Returns the northings of a specified contour generated by GenerateContours.
double *	**GetContourLong** (int contour) Returns the longitudes of a specified contour generated by GenerateContours.
double *	**GetContourEasting** (int contour) Returns the eastings of a specified contour generated by GenerateContours.
int	**GetNumGridY** (void) Return the number of grid points in the Y (north-south) dimension.
int	**GetNumGridX** (void) Return the number of grid points in the X (east-west) dimension.
double	**GetGridXY** (int xx, int yy) Return the value of the specified grid point.
double	**GetGridMax** (void) Return maximum value within the grid.
double	**GetGridMin** (void) Return minimum value within the grid.

3.1.1 Constructor & Destructor Documentation

ResultGrid::ResultGrid (ResultGrid * *theArray*)

This function is a copy constructor.

Parameters:

- *theArray* - the **ResultGrid** to be copied

```
ResultGrid::ResultGrid( double *      theData,
                        int           numRows,
                        int           numCols,
                        CDomain *     theDomain,
                        const char *  arrayName
                      )
```

This function instantiates a new **ResultGrid** with data contained in a linear double*array. Note: This function assumes the data points are located on the computational grid defined in *theDomain*. If they are not, the results are unpredictable.

Parameters:

- **array* - the returned data to be incorporated into the **ResultGrid**
- **theDomain* - the **CDomain** object associated the data returned by the m-file
- **arrayName* - the name to be associated with the data array

3.1.2 Member Function Documentation

```
void ResultGrid::GenerateContours( double contourValue )
```

This function generates a set of contours for the encapsulated data for the specified *contourValue*.

Parameters:

- *contourValue* - the contour value

```
double* ResultGrid::GetContourEasting( int contour )
```

This function returns the eastings of a specified contour generated by GenerateContours.

Parameters:

- *contour* - specifies the desired contour

Returns: An array of eastings for the points in the specified contour

```
double* ResultGrid::GetContourLat( int contour )
```

This function returns the latitudes of a specified contour generated by GenerateContours.

Parameters:

- *contour* - specifies the desired contour

Returns: An array of latitudes for the points in the specified contour

int ResultGrid::GetContourLength(int *contour*)

This function returns the length of a specified contour generated by **ResultGrid::GenerateContours**.

Parameters:

- *contour* - specifies the desired contour

Returns: The number of points in the specified contour

double* ResultGrid::GetContourLong(int *contour*)

This function returns the longitudes of a specified contour generated by GenerateContours.

Parameters:

- *contour* - specifies the desired contour

Returns: An array of longitudes for the points in the specified contour

double* ResultGrid::GetContourNorthing(int *contour*)

This function returns the northings of a specified contour generated by GenerateContours.

Parameters:

- *contour* - specifies the desired contour

Returns: An array of northings for the points in the specified contour

double ResultGrid::GetGridMax(void)

This function returns the maximum value within the grid.

Returns: The maximum grid value

double ResultGrid::GetGridMin(void)

This function returns the minimum value within the grid.

Returns: The minimum grid value

```
double ResultGrid::GetGridXY( int xx,
                              int yy
                            )
```

This function returns the value of the specified grid point.

Parameters:

- *xx* - X index of specified grid point
- *yy* - Y index of specified grid point

Returns: The value of the grid point with the specified indices

```
int ResultGrid::GetNumContours( void )
```

This function returns the number of contours generated by **ResultGrid::GenerateContours**.

Returns: The number of contours

```
int ResultGrid::GetNumGridX( void )
```

This function returns the number of grid points in the X (east-west) dimension.

Returns: The number of points in the X dimension

```
int ResultGrid::GetNumGridY( void )
```

This function returns the number of grid points in the Y (north-south) dimension.

Returns: The number of points in the Y dimension

3.2 SPEBE_Environment Class

The **SPEBE_Environment** class describes the general propagation environment of the region-of-interest, which encompasses the following:

- Domain (**SetDomain**)
  - coordinates of lower-left and upper-right corners of region-of-interest
  - time zone of lower-left corner
  - terrain elevations of region-of-interest
- Predominant ground type of region (**SetGroundType**, see also **SPEBE_GroundType**)
- Predominant seismological characteristics of region (**SetSeismicType**, see also **SPEBE_SeismicType**)
- Predominant meteorology of region (**SetMeteorology**, see also **TranslateCloudCover**).

Modify the enums **SPEBE_GroundType** and **SPEBE_SeismicType**, and the private methods GetGroundFileName and GetSeismicFileName, as necessary, to modify ground and seismic types.

A list of the public member functions for **SPEBE_Environment** class is given in table 3 and a list of the static member function for **SPEBE_Environment** class is given in table 4.

Table 3. Public member functions for the **SPEBE_Environment** class.

Function	Description
	SPEBE_Environment (void) Create a default environment, using the default files located under the directory defined in the ABFAHome environment variable.
	SPEBE_Environment (const char *envPath, const char *envName) Create environment saved from a previous SPEBE_Environment.
	~SPEBE_Environment (void) The destructor.
void	**SaveEnvironment** (const char *envPath, const char *envName) Save environment to a file.
void	**SetDomain** (double lowerLeftLat, double upperRightLat, double lowerLeftLon, double upperRightLon, const char *DTEDDirectory, int timeZone) Define domain: region coordinates, DTED directory for elevations, and timezone.
void	**SetDomain** (double lowerLeftLat, double upperRightLat, double lowerLeftLon, double upperRightLon, int timeZone) Define domain: region coordinates and timezone, with flat earth elevation model.
void	**SetGroundType** (**SPEBE_GroundType** groundType) Specify the predominant ground type.
void	**SetSeismicType** (**SPEBE_SeismicType** seismicType) Specify the predominant seismological characteristics.
void	**SetMeteorology** (double temp, double relativeHumidity, double windSpeed, double windDirection, int year, int dayOfYear, int timeOfDay, double cloudCover[3]) Specify the meteorological conditions for the domain.

Table 4. Static public member functions for the **SPEBE_Environment** class.

Function	Description
void	**TranslateCloudCover** (double cloudCover[3], const char *observation) Extract cloud cover from an observation string.

3.2.1 Constructor & Destructor Documentation

SPEBE_Environment::SPEBE_Environment(void)

This function creates a default environment, using the default files located under the directory defined in the ABFAHome environment variable.

Warning: These files are located in the Defaults subdirectory and are read-only. They should not be modified in any way.

```
SPEBE_Environment::SPEBE_Environment( const char * envPath,
                                      const char * envName
                                    )
```

This function creates the environment saved from a previous SPEBE_Environment.

Parameters:

- *envPath* - the directory path in which the environment file is located; don't include final \ If NULL, the ABFAHome directory will be used.
- *envName* - the name of the environment file; do''t include a file type

SPEBE_Environment::~SPEBE_Environment(void)

This function is the destructor.

Note: Always delete a SPEBE_Environment object when it is no longer needed.

3.2.2 Member Function Documentation

```
void SPEBE_Environment::SaveEnvironment( const char * envPath,
                                         const char * envName
                                       )
```

This function saves environment to a file.

Parameters:

- *envPath* - the directory path in which the environment file is to be stored; don't include final \. If NULL, the ABFAHome directory will be used.
- *envName* - the name of the environment file; don't include a file type

```
void SPEBE_Environment::SetDomain( double lowerLeftLat,
                                   double upperRightLat,
                                   double lowerLeftLon,
                                   double upperRightLon,
                                   int    timeZone
                                 )
```

This function defines domain: region coordinates and timezone with a flat earth elevation model.

Note: All coordinates are in signed decimal:

- + for east/north, - for west/south
- left of decimal represents degrees, right of decimal represents fraction of a degree

Parameters:

- *lowerLeftLat* - Latitude of the lower-left corner of the region-of-interest (in signed decimal)
- *upperRightLat* - Latitude of the upper-right corner of the region-of-interest (in signed decimal)
- *lowerLeftLon* - Longitude of the lower-left corner of the region-of-interest (in signed decimal)
- *upperRightLon* - Longitude of the upper-right corner of the region-of-interest (in signed decimal)
- *timeZone* - the difference between local time of the lower-left corner and Greenwich Mean Time (GMT)

```
void SPEBE_Environment::SetDomain( double        lowerLeftLat,
                                   double        upperRightLat,
                                   double        lowerLeftLon,
                                   double        upperRightLon,
                                   const char *  DTEDDirectory,
                                   int           timeZone
                                 )
```

This function defines domain: region coordinates, Digital Terrain Elevation Data (DTED) directory for elevations, and timezone.

Note: All coordinates are in signed decimal:

- + for east/north, - for west/south
- left of decimal represents degrees, right of decimal represents fraction of a degree

Parameters:

- *lowerLeftLat* - Latitude of the lower-left corner of the region-of-interest (in signed decimal)
- *upperRightLat* - Latitude of the upper-right corner of the region-of-interest (in signed decimal)
- *lowerLeftLon* - Longitude of the lower-left corner of the region-of-interest (in signed decimal)
- *upperRightLon* - Longitude of the upper-right corner of the region-of-interest (in signed decimal)
- *DTEDDirectory* - a directory containing DTED data, including a file named DMED and a DTED subdirectory. This can be a DTED CD or directory on hard drive, flash disk, etc.
- *timeZone* - the difference between local time of the lower-left corner and GMT

```
void SPEBE_Environment::SetGroundType( SPEBE_GroundType groundType )
```

This function specifies the predominant ground type.

Parameters:

- *groundType* - The ground type, chosen from **SPEBE_GroundType**

```
void SPEBE_Environment::SetMeteorology( double temp,
                                        double relativeHumidity,
                                        double windSpeed,
                                        double windDirection,
                                        int    year,
                                        int    dayOfYear,
                                        int    timeOfDay,
                                        double cloudCover[3]
                                      )
```

This function specifies the meteorological conditions for the domain.

Parameters:

- *temp* - surface temperature in degrees Celsius
- *relativeHumidity* - the surface relative humidity in percent
- *windSpeed* - the surface wind speed in meters per second
- *windDirection* - the surface wind direction in degrees (counterclockwise from east?)
- *year* - the full four digit Gregorian calendar year
- *dayOfYear* - the day of the year starting from 1 for January 1
- *timeOfDay* - the time of day in 24-h format starting at midnight = 0.
- *cloudCover* - a three-element array giving the fractional cloud cover at low, medium, and high elevation. See **TranslateCloudCover**.

```
void SPEBE_Environment::SetSeismicType( SPEBE_SeismicType seismicType )
```

This function specifies the predominant seismological characteristics.

Parameters:

- *seismicType* - The seismic type, chosen from **SPEBE_SeismicType**

void SPEBE_Environment::TranslateCloudCover(double *cloudCover*[3], const char * *observation*) `[static]`

This function extracts cloud cover from an observation string.

Parameters:

- *observation* - a National Weather Service local observation string
- *cloudCover* - a three-element array, allocated in the calling program, returning the fractional cloud cover at low, medium, and high elevation.

3.3 SPEBE_PDCalculator Class

The **SPEBE_PDCalculator** class generates detection probabilities for a *fixed*

- environment (region and weather: **SPEBE_Environment**);
- background noise level (**SPEBE_NoiseType**);
- target type (**SPEBE_TargetType**), height, and direction; and
- set of sensors (**SPEBE_SensorSet**).

The **SPEBE_PDCalculator** allows individual sensors to be activated and deactivated, and returns detection probabilities for the set of activated sensors, or for any specified sensor regardless of its activation state. Note: All sensors are initially *activated*.

Attention: SPEBE_PDCalculator works on a snapshot of the constructor arguments, as does the **SPEBE_PropagationTable**. Thus, any changes to the **SPEBE_SensorSet**, such as location of a particular sensor, will require instantiation of a new **SPEBE_PDCalculator**. Likewise, any changes to the environment require a new **SPEBE_PropagationTable** and in turn a new **SPEBE_PDCalculato**r.

The public member functions for the **SPEBE_PDCalculator** class are listed in table 5.

Table 5. Public member functions for the **SPEBE_PDCalculator** class.

Function	Description
	SPEBE_PDCalculator (**SPEBE_PropagationTable** *table, **SPEBE_SensorSet** *sensorSet, **SPEBE_TargetType** targetType, **SPEBE_NoiseType** noiseType) Constructs a PDCalculator for the specified **SPEBE_PropagationTable**, **SPEBE_SensorSet**, **SPEBE_TargetType**, and **SPEBE_NoiseType**, using the default target height.
	SPEBE_PDCalculator (**SPEBE_PropagationTable** *table, **SPEBE_SensorSet** *sensorSet, **SPEBE_TargetType** targetType, double targetHeight, double targetDirection, **SPEBE_NoiseType** noiseType) Constructs a PDCalculator for the specified **SPEBE_PropagationTable**, **SPEBE_SensorSet**, **SPEBE_TargetType**, and **SPEBE_NoiseType**, using a specified target height.
	~SPEBE_PDCalculator (void) The destructor.
ResultGrid *	**GetPD** (void) Return the detection probability grid for the active sensors, as determined by **Activate** and **Deactivate**.
ResultGrid *	**GetPD** (int rcvrIndex) Return the detection probability grid for the active sensors, as determined by **Activate** and **Deactivate**, and specified target height Returns the detection probability grid for the specified sensor, regardless of active or inactive state.
void	**Activate** (int rcvrIndex) Returns the detection probability grid for the specified sensor and target height, regardless of active or inactive state. Activates the specifed sensor.
void	**Deactivate** (int rcvrIndex) Deactivates the specifed sensor.

3.3.1 Constructor & Destructor Documentation

```
SPEBE_PDCalculator::SPEBE_PDCalculator( SPEBE_PropagationTable * table,
                                        SPEBE_SensorSet *        sensorSet,
                                        SPEBE_TargetType         targetType,
                                        SPEBE_NoiseType          noiseType
                                      )
```

This function constructs a PDCalculator for the specified **SPEBE_PropagationTable**, **SPEBE_SensorSet**, **SPEBE_TargetType**, and **SPEBE_NoiseType**, using the default target height.

Parameters:

- *table* - The **SPEBE_PropagationTable** describing the propagation environment and file prefix for the propagation tables to be used in the detection probability calculations.
- *sensorSet* - The **SPEBE_SensorSet** describing the set of sensors for which detection probability calculations we be performed.
- *targetType* - The target type, chosen from **SPEBE_TargetType**
- *noiseType* - The background noise type, chosen from **SPEBE_NoiseType**

```
SPEBE_PDCalculator::SPEBE_PDCalculator( SPEBE_PropagationTable * table,
                                        SPEBE_SensorSet *        sensorSet,
                                        SPEBE_TargetType         targetType,
                                        double                   targetHeight,
                                        double                   targetDirection,
                                        SPEBE_NoiseType          noiseType
                                       )
```

This function constructs a PDCalculator for the specified **SPEBE_PropagationTable**, **SPEBE_SensorSet**, **SPEBE_TargetType**, and **SPEBE_NoiseType**, using a specified target height.

Parameters:

- *table* - The **SPEBE_PropagationTable** describing the propagation environment and file prefix for the propagation tables to be used in the detection probability calculations.
- *sensorSet* - The **SPEBE_SensorSet** describing the set of sensors for which detection probability calculations we be performed.
- *targetType* - The target type, chosen from **SPEBE_TargetType**
- *targetHeight* - The target height, in meters
- *noiseType* - The background noise type, chosen from **SPEBE_NoiseType**

```
SPEBE_PDCalculator::~SPEBE_PDCalculator( void  )
```

This function is the destructor.

Note: Always delete a SPEBE_PDCalculator object when it is no longer needed.

3.3.2 Member Function Documentation

```
void SPEBE_PDCalculator::Activate( int rcvrIndex )
```

This function returns the detection probability grid for the specified sensor and target height, regardless of active or inactive state, and activates the specified sensor.

Parameters:

- *rcvrIndex* - the receiver to be activated; indexing starts at zero.

void SPEBE_PDCalculator::Deactivate(int *rcvrIndex*)

This function deactivates the specified sensor.

Parameters:

- *rcvrIndex* - the receiver to be deactivated; indexing starts at zero.

ResultGrid * SPEBE_PDCalculator::GetPD(int *rcvrIndex*)

This function returns the detection probability grid for the active sensors, as determined by **Activate** and **Deactivate**, and specified target height returns the detection probability grid for the specified sensor, regardless of active or inactive state.

Parameters:

- *rcvrIndex* - the requested receiver; indexing starts at zero.

Returns: A **ResultGrid** object containing the requested detection probability data

ResultGrid * SPEBE_PDCalculator::GetPD(void)

This function returns the detection probability grid for the active sensors, as determined by **Activate** and **Deactivate**. **NOTE:** All sensors are initially *deactivated*.

Returns: A **ResultGrid** object containing the requested detection probability data

3.4 SPEBE_PropagationTable Class

The **SPEBE_PropagationTable** class generates and manages the propagation tables for a given environment (**SPEBE_Environment**). These tables are used by the **SPEBE_PDCalculator** to calculate detection probabilities. If you want to avoid errors leading to invalid results, **make sure you read and understand the documentation** for this class.

In addition to generating the propagation tables for a given environment and specified target and sensor types (**SPEBE_TargetType**, **SPEBE_SensorType**), **SPEBE_PropagationTable** saves them under a unique filename in a specified path. If no path is given in the constructor, then the path given by the environment variable ABFAHome will be used.

Two propagation models are available for the calculations: a simple but fast spherical spreading model with ground impedance and a full-wave, split-step parabolic equation. The spherical spreading will account for ground reflection and the spreading of the acoustic wavefront as it expands away from the source, but it will ignore refractive effects of the atmosphere. While this model is very fast, it may give poor results in many cases because the refraction effects are generally significant. The parabolic equation will include all the effects of the spherical spreading model along with refraction, thus giving much more accurate results. However, it requires considerably more time to complete the calculations.

The spherical spreading model gives reasonable results for a source more than a few hundred meters above the ground, because most of the refractive effects are close to the ground. For sources closer to the ground, the parabolic equation will give much better results. To use the parabolic equation below a specified height, call **EnableHiFi(double height)**. This will automatically invoke the spherical spreading above the specified height and the parabolic equation below that height. To use the spherical spreading model exclusively, regardless of height, call **DisableHiFi(void)**.

In either case, for each table generated, a source and sensor height label will be appended to the file prefix specified in the constructor, and the table will be saved in the file of that name, in the directory specified in the constructor. Thus, a set of tables will be generated and stored in files, depending on the environment and types of sources and sensors specified.

Only the source and sensor heights are important, so a new table may not be generated for each call to **GenerateTable**, depending upon the heights of the specified source and sensor types and the heights associated with any tables that may have already been generated.

You do not have to explicitly generate any propagation tables (but you do need to instantiate a **SPEBE_PropagationTable** object). The **SPEBE_PDCalculator** will generate tables as needed. However, you can generate them if you prefer, so that once you start making detection probability calculations, you won't have any long delays as new tables are generated.

Note: SPEBE_PropagationTable uses a snapshot of the specified **SPEBE_Environment**. If any changes are made to the environment after instantiation of the SPEBE_PropagationTable, they will not be incorporated. Thus, a new SPEBE_PropagationTable must be instantiated for the modified environment. The intent is that a given SPEBE_PropagationTable represents a given environmental state, and **different environmental states should be associated with different filenames**.

Attention: Propagation tables are permanently saved for reuse. If propagation table files with the specified prefix, in the specified directory, already exist, they will be reused. This allows tables generated in a previous session to be reused without recomputing, but also calls for careful use of file prefix naming and table housekeeping. **Be sure to read the warnings at the bottom of this section.**

Reusing Tables and/or File Prefixes

To reuse tables at a later session, use one of two options:

- First option:
    1. Before terminating a session, save the **SPEBE_PropagationTable** (which includes the associated environment) using **SPEBE_PropagationTable::SaveEnvironment(void)**.
    2. In the new session, generate a new **SPEBE_PropagationTable** using **SPEBE_PropagationTable::SPEBE_PropagationTable(char* tablePath, char* tablePrefix)** with the same tablePath and tablePrefix as was used in the previously saved **SPEBE_PropagationTable**.
- Second option:
    1. Before terminating the session, save the **SPEBE_Environment** used to generate the **SPEBE_PropagationTable**, using S**PEBE_Environment::SaveEnvironment(char* envPath, char* envName)** with any desired envPath and envName.
    2. In the new session, generate a new **SPEBE_Environment** using **SPEBE_Environment::SPEBE_Environment(char* envPath, char* envName)** with the same envPath and envName used to save in the previous session.
    3. Then, generate a new **SPEBE_PropagationTable** using **SPEBE_PropagationTable::SPEBE_PropagationTable(SPEBE_Environment* theEnvironment, char* tablePath, char* tablePrefix)** using the **SPEBE_Environment** just generated, and the same tablePath and tablePrefix used by the **SPEBE_PropagationTable** in the previous session.

Warning: The second option will give invalid results if changes were made to the **SPEBE_Environment** after the **SPEBE_PropagationTable** was instantiated.

In general, if a **SPEBE_PropagationTable** is instantiated with a previously used tablePath and tablePrefix, and a **SPEBE_Environment** that does not represent the environmental state of the previously generated table files, the results will be invalid.

Thus, exercise caution when reusing previously generated tables, or when reusing an old tablePath and tablePrefix to generate new files.

In particular, if a file directory and prefix are to be reused to generate new tables, then the old tables must first be deleted. It is also advisable to delete old tables that are no longer needed.

The public member functions for the **SPEBE_PropagationTable** class are listed in table 6.

Table 6. Public member functions for the SPEBE_PropagationTable class.

Function	Description
	SPEBE_PropagationTable (**SPEBE_Environment** *environment, const char *tablePath, const char *tablePrefix) Construct a new SPEBE_PropagationTable for the specified environment, tablePath, and tablePrefix.
	SPEBE_PropagationTable (const char *tablePath, const char *tablePrefix) Construct a new SPEBE_PropagationTable from a previously saved **SPEBE_Environment** and SPEBE_PropagationTable using **SaveEnvironment**.
	~SPEBE_PropagationTable (void) The destructor.
void	**SaveEnvironment** (void) Save the SPEBE_PropagationTable and associated **SPEBE_Environment** under the same tablePrefix as the tables, for later reuse.
void	**EnableHiFi** (double height) Enable the use of high-fidelity propagation modeling for near-ground sources.
void	**DisableHiFi** (void) Disable the use of high-fidelity propagation modeling for near-ground sources.
bool	**HiFi** (double height) Indicate whether high-fidelity propagation modeling will be used for specified height.
void	**GenerateTable** (**SPEBE_SensorType** sensorType, **SPEBE_TargetType** targetType) Construct and save a table for the given sensor and target types, using the default target height.
void	**GenerateTable** (**SPEBE_SensorType** sensorType, double targetHt) Construct and save a table for the given sensor type and target height.
const char *	**GetTableFileName** (**SPEBE_SensorType** sensorType, **SPEBE_TargetType** targetType) Get the full name of the propagation table file for the specified sensor and target type, using the default target height. /note These are .mat files and are managed by the API internally. Thus, their direct manipulation is not required.
const char *	**GetTableFileName** (**SPEBE_SensorType** sensorType, double targetHt) Get the full name of the propagation table file for the specified sensor type and target height. /note These are .mat files and are managed by the API internally. Thus, their direct manipulation is not required.

3.4.1 Constructor & Destructor Documentation

```
SPEBE_PropagationTable::SPEBE_PropagationTable( SPEBE_Environment * environment,
                                                const char *        tablePath,
                                                const char *        tablePrefix
                                              )
```

This function constructs a new **SPEBE_PropagationTable** for the specified environment, tablePath, and tablePrefix.

Parameters:

- *tablePath* - the directory path in which the propagation tables will be placed; don't include final slash "\"
- *tablePrefix* - the file prefix to be used in naming the propagation tables; don't include a file type

```
SPEBE_PropagationTable::SPEBE_PropagationTable( const char * tablePath,
                                                const char * tablePrefix
                                              )
```

This function constructs a new SPEBE_PropagationTable from a previously saved SPEBE_Environment and SPEBE_PropagationTable using SaveEnvironment.

Parameters:

- *tablePath* - the directory path in which the environment and table files are located; don't include a final directory slash "\"
- *tablePrefix* - the file prefix of the previously saved **SPEBE_PropagationTable**; don't include a file type

```
SPEBE_PropagationTable::~SPEBE_PropagationTable( void )
```

This function is the destructor.

Note: Always delete a **SPEBE_PropagationTable** object when it is no longer needed.

3.4.2 Member Function Documentation

void SPEBE_PropagationTable::DisableHiFi(void)

This function disables the use of high-fidelity propagation modeling for near-ground sources.

Warning: This must be enable or disabled PRIOR to the generation of any propagation tables. If it is changed after propagation tables have been generated, then the tables will not be regenerated, i.e., the new value will be ignored.

void SPEBE_PropagationTable::EnableHiFi(double *height*)

This function enables the use of high-fidelity propagation modeling for near-ground sources.

Warning: This must be enable or disabled PRIOR to the generation of any propagation tables. If it is changed after propagation tables have been generated, then the tables will not be regenerated, i.e., the new value will be ignored.

Parameters:

- *height* - Specify the height below which the high-fidelity will be used.

void SPEBE_PropagationTable::GenerateTable(SPEBE_SensorType *sensorType*, double *targetHt*)

This function constructs and saves a table for the given sensor type and target height. The table will be saved in the directory specified in the constructor. The filename prefix will be prepended to a coded height string to arrive at the final filename.

Warning: If the file already exists, it will not be generated again. This results in efficient use of tables for calculations, but requires care in the use of file prefixes. See comments under the description of the class.

Parameters:

- *sensorType* - The sensor type, chosen from **SPEBE_SensorType**
- *targetHt* - The target height

```
void SPEBE_PropagationTable::GenerateTable( SPEBE_SensorType sensorType,
                                            SPEBE_TargetType targetType
                                          )
```

This function constructs and saves a table for the given sensor and target types, using the default target height. The table will be saved in the directory specified in the constructor. The filename prefix will be prepended to a coded height string to arrive at the final filename.

Warning: If the file already exists, it will not be generated again. This results in efficient use of tables for calculations, but requires care in the use of file prefixes. See comments under the description of the class.

Parameters:

- *sensorType* - The sensor type, chosen from **SPEBE_SensorType**
- *targetType* - The target type, chosen from **SPEBE_TargetType**

```
const char * SPEBE_PropagationTable::GetTableFileName( SPEBE_SensorType sensorType,
                                                       double           targetHt
                                                     )
```

This function gets the full name of the propagation table file for the specified sensor type and target height. Note: These are *.mat* files and are managed by the API internally. Thus, their direct manipulation is not required.

Parameters:

- *sensorType* - The sensor type, chosen from **SPEBE_SensorType**
- *targetHt* - The target height

```
const char * SPEBE_PropagationTable::GetTableFileName( SPEBE_SensorType sensorType,
                                                       SPEBE_TargetType targetType
                                                     )
```

This function gets the full name of the propagation table file for the specified sensor and target type, using the default target height. NOTE: These are *.mat* files and are managed by the API internally. Thus, their direct manipulation is not required.

Parameters:

- *sensorType* - The sensor type, chosen from **SPEBE_SensorType**
- *targetType* - The target type, chosen from **SPEBE_TargetType**

void SPEBE_PropagationTable::SaveEnvironment(void)

This function saves the **SPEBE_PropagationTable** and associated **SPEBE_Environment** under the same tablePrefix as the tables for later reuse.

Note: Individual propagation tables are saved automatically as they are generated. This method saves the **SPEBE_PropagationTable** and associated **SPEBE_Environment** for reuse.

3.5 SPEBE_SensorSet Class

The **SPEBE_SensorSet** class manages a set of sensors chosen from **SPEBE_SensorType**. A sensor set is instantiated with a specified number of sensors of identical type. Coordinates of each sensor are then set individually. If any sensor is of a different type, it can be changed individually using **ChangeSensorType**; the assumption is that in most cases there will be a predominant type.

Note: Sensors cannot be added or deleted once the set is created; however, sensor types and coordinates can be changed. Also, specific sensors can be activated and deactivated for detection probability calculations (see **SPEBE_PDCalculator**).

Warning: If you change the type of any of the sensors, a new set of parameters will be set up for that sensor, including the coordinates. Thus, the coordinates of each sensor must be set after the type is specified.

The public member functions for **SPEBE_SensorSet** class are listed in table 7 and the static public member functions for this class are listed in table 8.

Table 7. Public member functions for the **SPEBE_SensorSet** class.

Function	Description
	SPEBE_SensorSet (int numSensors, **SPEBE_SensorType** sensorType) Instantiate a specfied number of specified sensor type.
	SPEBE_SensorSet (**SPEBE_SensorSet** *sensorSet) Copy constructor.
	~SPEBE_SensorSet (void) The destructor.
void	**ChangeSensorType** (int sensorNumber, **SPEBE_SensorType** sensorType) Change the sensor type of the specified sensor. Indexing starts with zero.
SPEBE_SensorType	**GetSensorType** (int sensorNumber) Indicate the sensor type of the specified sensor.
int	**GetNumSensors** (void) Indicate the number of sensors.
void	**SetSensorLocation** (int sensorNumber, double lat, double lon) Set the location of the specified sensor. Indexing starts with zero.
double	**GetSensorLat** (int sensorNumber) Indicate the specified sensor latitude.
double	**GetSensorLon** (int sensorNumber) Indicate the specified sensor longitude.

Table 8. Static public member functions for the **SPEBE_SensorSet** class.

Function	Description
double	**GetSensorHeight** (**SPEBE_SensorType** sensorType) Indicate the specified sensor height.

3.5.1 Constructor & Destructor Documentation

SPEBE_SensorSet::SPEBE_SensorSet(int ***numSensors,***
SPEBE_SensorType ***sensorType***
)

This function instantiates a specified number of specified sensor type. Once instantiated, particular sensor types can be changed using **ChangeSensorType**.

Parameters:

- *numSensors* - the number of sensors in the set.
- *sensorType* - The sensor type, chosen from **SPEBE_SensorType**

SPEBE_SensorSet::SPEBE_SensorSet(SPEBE_SensorSet * ***sensorSet*** **)**

This function is a copy constructor. Once instantiated, particular sensor types can be changed using **ChangeSensorType**.

Parameters:

- *sensorSet* - the existing **SPEBE_SensorSet** to be copied.

SPEBE_SensorSet::~SPEBE_SensorSet(void)

This function is the destructor.

Parameters: *none*

Note: Always delete a **SPEBE_SensorSet** object when it is no longer needed.

3.5.2 Member Function Documentation

void SPEBE_SensorSet::ChangeSensorType(int ***sensorNumber,***
SPEBE_SensorType ***sensorType***
)

This function changes the sensor type of the specified sensor. Indexing starts with zero.

Parameters:

- *sensorNumber* - the particular sensor within the set to be changed; indexing starts at zero.
- *sensorType* - The sensor type, chosen from **SPEBE_SensorType**

int SPEBE_SensorSet::GetNumSensors(void)

This function indicates the number of sensors.

Returns: the number of sensors

double SPEBE_SensorSet::GetSensorHeight(SPEBE_SensorType ***sensorType*** **)** `[static]`

This function indicates the specified sensor height.

Parameters:

- *sensorNumber* - the particular sensor within the set. Indexing starts at zero.

Returns: The specified sensor height

double SPEBE_SensorSet::GetSensorLat(int ***sensorNumber*** **)**

This function indicates the specified sensor latitude.

Parameters:

sensorNumber - the particular sensor within the set; indexing starts at zero.

Returns: The specified sensor latitude

double SPEBE_SensorSet::GetSensorLon(int *sensorNumber*)

This function indicates the specified sensor longitude.

Parameters:

- *sensorNumber* - the particular sensor within the set; indexing starts at zero.

Returns: The specified sensor longitude

SPEBE_SensorType SPEBE_SensorSet::GetSensorType(int *sensorNumber*)

This function indicates the sensor type of the specified sensor.

Parameters:

- *sensorNumber* - the particular sensor within the set to be changed; indexing starts at zero.

Returns: The specified sensor type

```
void SPEBE_SensorSet::SetSensorLocation( int      sensorNumber,
                                         double lat,
                                         double lon
                                       )
```

This function sets the location of the specified sensor. Indexing starts with zero. NOTE: This must be called after **ChangeSensorType** if the sensor type is to be changed.

Parameters:

- *sensorNumber* - the particular sensor within the set to be changed; indexing starts at zero
- *lat* - the latitude of the sensor, in signed decimal
- *lat* - the longitude of the sensor, in signed decimal

3.6 SPEBE_TargetSet Class

The **SPEBE_TargetSet** class manages a set of targets, or acoustic/seismic sources, chosen from **SPEBE_TargetType**. A target set is instantiated with a specified number of targets of identical type. Coordinates of each target are then set individually. If any targets are of a different type, they can be changed individually using **ChangeTargetType**; the assumption is that in most cases there will be a predominant type.

Note: Targets cannot be added or deleted once the set is created; however, target types and coordinates can be changed. Also, specific targets can be activated and deactivated for detection probability calculations (see **SPEBE_PDCalculator**).

Warning: If you change the type of any of the targets, a new set of parameters will be set up for that target, including the coordinates. Thus, the coordinates of each target must be set after the type is specified.

The public member functions for the SPEBE_TargetSet Class are listed in table 9 and the static public member function is listed in table 10.

Table 9. Public member functions for the **SPEBE_TargetSet** class.

Function	Description
	SPEBE_TargetSet (int numTargets, **SPEBE_TargetType** targetType) Instantiate a specfied number of specified target type.
	~SPEBE_TargetSet (void) The destructor.
void	**ChangeTargetType** (int targetNumber, **SPEBE_TargetType** targetType) Change the target type of the specified target. Indexing starts with zero.
SPEBE_TargetType	**GetTargetType** (int targetNumber) Indicate the specified target type.
int	**GetNumTargets** (void) Indicate the number of targets in the set.
void	**SetTargetLocation** (int targetNumber, double lat, double lon) Set the location of the specified target. Indexing starts with zero.
void	**SetTargetHeight** (int targetNumber, double height) Set the height of the specified target. Indexing starts with zero.
void	**SetTargetDirection** (int targetNumber, double dir) Set the direction of the specified target. Indexing starts with zero.
double	**GetTargetLat** (int targetNumber) Indicate the specified target latitude.
double	**GetTargetLon** (int targetNumber) Indicate the specified target longitude.

Table 10. Static public member functions for the **SPEBE_TargetSet** class.

Function	Description
double	**GetTargetHeight** (**SPEBE_TargetType** targetType) Indicate the specified target height.

3.6.1 Constructor & Destructor Documentation

SPEBE_TargetSet::SPEBE_TargetSet(int ***numTargets,***
SPEBE_TargetType ***targetType***
)

This function instantiates a specified number of specified target types. Once instantiated, particular target types can be changed using **ChangeTargetType**.

Parameters:

- *numTargets* - the number of targets in the set.
- *targetType* - The target type, chosen from **SPEBE_TargetType**

SPEBE_TargetSet::~SPEBE_TargetSet(void)

This function is the destructor. **NOTE:** Always delete a **SPEBE_TargetSet** object when it is no longer needed.

3.6.2 Member Function Documentation

void SPEBE_TargetSet::ChangeTargetType(int ***targetNumber,***
SPEBE_TargetType ***targetType***
)

This function changes the target type of the specified target. Indexing starts with zero.

Parameters:

- *targetNumber* - the particular target within the set to be changed; indexing starts at zero.
- *targetType* - The target type, chosen from **SPEBE_TargetType**

int SPEBE_TargetSet::GetNumTargets(void)

This function indicates the number of targets in the set.

Returns: The number of targets

double SPEBE_TargetSet::GetTargetHeight(SPEBE_TargetType ***targetType*** **)** `[static]`

This function indicates the specified target height.

Parameters:

- *sensorNumber* - the particular target within the set; indexing starts at zero.

Returns: The specified target height, in meters above ground

double SPEBE_TargetSet::GetTargetLat(int *targetNumber*)

This function indicates the specified target latitude.

Parameters:

- *sensorNumber* - the particular target within the set; indexing starts at zero.

Returns: The specified target latitude

double SPEBE_TargetSet::GetTargetLon(int *targetNumber*)

This function indicates the specified target longitude.

Parameters:

- *sensorNumber* - the particular target within the set; indexing starts at zero.

Returns: The specified target longitude

SPEBE_TargetType SPEBE_TargetSet::GetTargetType(int *targetNumber*)

This function indicates the specified target type.

Parameters:

- *sensorNumber* - the particular target within the set; indexing starts at zero.

Returns: The specified target type

```
void SPEBE_TargetSet::SetTargetDirection( int    targetNumber,
                                          double dir
                                        )
```

This function sets the direction of the specified target. Indexing starts with zero. Note: This refers to the direction the target is pointing, not the direction of the target's ground track. In the case of an aircraft flying with a crosswind component, the two will not be the same. This must be called after **ChangeTargetType** if the target type is to be changed.

Parameters:

- *targetNumber* - the particular target within the set; indexing starts at zero.
- *dir* - the direction the target is pointing, in degrees

```
void SPEBE_TargetSet::SetTargetHeight( int       targetNumber,
                                       double height
                                     )
```

This function sets the height of the specified target. Indexing starts with zero. Note: This must be called after **ChangeTargetType** if the target type is to be changed.

Parameters:

- *targetNumber* - the particular target within the set; indexing starts at zero.
- *height* - the height above ground of the sensor, in meters

```
void SPEBE_TargetSet::SetTargetLocation( int       targetNumber,
                                         double lat,
                                         double lon
                                       )
```

This function sets the location of the specified target. Indexing starts with zero. Note: This must be called after **ChangeTargetType** if the target type is to be changed.

Parameters:

- *targetNumber* - the particular target within the set to be changed; indexing starts at zero.
- *lat* - the latitude of the sensor, in signed decimal
- *lat* - the longitude of the sensor, in signed decimal

4. Acoustic Sensor Placement Global Definitions

This section includes the definitions of several global enumerated types and functions.

The enumerated types are used by various class methods to specify predefined choices for ground, seismic, sensor, target, and noise types. Each determines a set of parameters defined in the data files that are included with the SPEBE and ASSP distribution.

The functions are associated with operation of the overall API rather than any specific class, and are therefore defined independently of the classes.

4.1 Global Enumerated Types

4.1.1 Enumeration Type Documentation

enum SPEBE_GroundType

This lists the ground types for environmental characterization. The types are Urban, Suburban, Asphalt, Gravel, Sand, Brush, Forest, ShortGrass, LongGrass, OpenWater, Ice, and Snow. See also **SPEBE_Environment**.

enum SPEBE_NoiseType

This lists the background noise types for detection probability calculations. The types are Rural, City, LightBattle, and IntenseBattle.

enum SPEBE_SeismicType

This lists the seismic types for environmental characterization. The types are Desert, SoilOverBedrock, and SiltOverWaterTable.

enum SPEBE_SensorType

This lists the sensor types for detection probability calculations and propagation table generation. The types are Human, Acoustic, and Seismic, where human refers to a person with International Standards Organization (ISO) Standard good hearing, acoustic refers to a generic three-element microphone array, and seismic refers to a generic geophone.

enum SPEBE_TargetType

This lists the target types for detection probability and propagation table generation. The types are Tracked, WheeledHeavy, WheeledLight, HelicopterLow, HelicopterHigh, T2_UAS, T3_UAS, and Personnel. T2_UAS refers to a Shadow surrogate Unmanned Arial System (UAS), T3_UAS refers to a Hunter surrogate UAS, and Personnel refers to the seismic signature of adult human footsteps.

4.2 Global Functions

4.2.1 Function Documentation

void SPEBE_Initialize(void)

This function shows the initialization to be called once at the beginning of a session.

Parameters: *none*

void SPEBE_Shutdown(void)

This is the function to be called once at the end of a session.

Parameters: *none*

5. Conclusion

This API was used by the U.S. Army Communications-Electronics Research, Development and Engineering Center (CERDEC), Command and Control Directorate (C2D), to provide acoustic sensor capability to graphical user interface (GUI)-based multimode sensor placement decision aid development projects. Much of the architecture and interface design was based on requirements provided by C2D, which in turn were derived from operational requirements for the deployment of acoustic sensors and sensor decision aids. It was implemented by C2D developers with general expertise in GUI and decision aid development but limited expertise in acoustic sensors. The relative ease with which these developers were able to integrate ASSP into their software demonstrates the success of the API as a tool to enable the development of acoustic sensor placement decision aids with minimal acoustics expertise.

Acronyms

2-D	two-dimensional
API	Application Programmer Interface
C2D	Command and Control Directorate
CERDEC	Communications-Electronics Research, Development and Engineering Center
DTED	Digital Terrain Elevation Data
GMT	Greenwich Mean Time
GUI	graphical user interface
ISO	International Standards Organization
JNI	Java Native Interface
NSFF	Networked Sensors for the Future Force
PD	probability of detection
SPEBE	Sensor Performance Evaluator for Battlefield Environments
SPEBE4NSFF	SPEBE for NSFF
UAS	Unmanned Arial System

No. of Copies	Organization
1 (PDF ONLY)	ADMNSTR
	DEFNS TECHL INFO CTR
	DTIC OCP (ELECTRONIC COPY)
	8725 JOHN J KINGMAN RD STE 0944
	FT BELVOIR VA 22060-6218
3 HCs	US ARMY RSRCH LAB
	IMNE ALC IMS MAIL & RECORDS MGMT
	AMSRD ARL CI OK TL TECHL LIB
	AMSRD ARL CI OK T TECHL PUB
	2800 POWDER MILL ROAD
	ADELPHI MD 20783-1197
1 HC	US ARMY RESEARCH LAB
	AMSRD CI OK TP TECHL LIB
	ATTN T LANDFRIED
	APG MD 21005
4 HCs	US ARMY RESEARCH LAB
	AMSRD ARL CI EM
	L PARKER
	AMSRD ARL CI ES
	DR J NOBLE
	DR S COLLIER
	AMSRD ARL SE SE
	N SROUR
	2800 POWDER MILL ROAD
	ADELPHI MD 20783-1197

No. of Copies	Organization
1 HC	US ARMY RESEARCH LAB
	AMSRD ARL CI EM
	DR D MARLIN
	WSMR NM 88002-5501
1 HC	ERDC CRREL
	SIGNATURE PHYSICS BRANCH
	DR K WILSON
	72 LYME RD
	HANOVER NH 03755-1290
1 HC	US ARMY ARDEC
	AMSTA AR FSF R
	J CHANG
	BLDG 95N
	PICATINNY ARSENAL NJ 07806-5000
1 HC	RDECOM ARDEC
	AMSRD AAR AEP A
	J HEBERLEY
	BLDG 407
	PICATINNY ARSENAL NJ 07806-5000
Total:	1 PDF, 12 HCs